1.3 No 5 & 6 : Modèle proie-prédateur

« Parmi les biologues [sic] et les physiciens d'une grande renommée jouissent les travaux de Lokta (1923) et de Volterra (1926), d'après lesquels la destruction d'une espèce (la victime) par l'autre (l'agresseur, le parasite) peut amener des fluctuations périodiques de la population des deux epèces. Cette théorie est fondée sur certaines hypothèses des plus simples concernant l'action réciproque des agresseurs (N_2) et des victimes (N_1) qui ont été formulées comme une équation différentielle

$$\begin{array}{rcl} \frac{dN_1}{dt} & = & b_1\ N_1 - k_1\ N_1\ N_2 \\ \frac{dN_2}{dt} & = & k_2\ N_1\ N_2 - d_2\ N_2 \end{array}$$

Ici $b_1\ N_1$ désigne l'accroissement géométrique des victimes, $k_1\ N_1\ N_2$ la destruction des victimes par les agresseurs et $d_2\ N_2$ la mortalité de ceux-ci. » Vérification expérimentales de la théorie mathématique de la lutte pour la vie, G.F. Gause, Hermann, Paris, 1935.

Verifions tout cela.

Question 1 *Tracer le portrait de phase avec la fonction (N_1 et N_2 sont devenus x et y)*

```
load("plotdf");
plotdf([b1*x-k1*x*y,k2*x*y-d2*y],[x,y],
[parameters,"b1=1,k1=1,d2=1,k2=1"],
[tstep,0.1],[nsteps,100],
[direction,forward],[x,0,10],[y,0,10])$
```

Comprendre chaque argument de la fonction plotdf. Que représentent les vecteurs tracés ? Que valent-ils en $x = 0$ puis en $y = 0$?

Les vecteurs tracés sont ceux de coordonnées $(x\ (1-y), -y\ (1-x))$; ils représentent la direction vers laquelle évolue le point de coordonnées (x, y). Ils sont même les tangentes des lignes solutions $x(t), y(t)$ et de ce point de vue la résolution d'un système différentiel peut être vue comme la recherche des lignes dont les tangents sont des données.

En $x = 0$ ces vecteurs sont de la forme $(0, -y)$ et en $y = 0$ de la forme $(x, 0)$; et donc tout point de coordonnées (x, y) qui se trouverait à un instant en $x = 0$ ou $y = 0$ continuerait à se déplacer sur la droite $x = 0$ ou $y = 0$.

Question 2 *L'intégration à partir d'une position initiale de ce système autonome se lance en cliquant sur cette position dans la fenêtre. Quelle est la forme des solutions ? Attention, il ne faut pas prêter trop d'attention à la valeur du pas de temps tstep qui ne semble pas actif et fixé à 0.1 quoi qu'on fasse ; plutôt jouer sur nsteps le nombre de pas de temps.*

À partir du point $(4, 1)$

```
 plotdf([b1*x-k1*x*y,k2*x*y-d2*y],[x,y],
[parameters,"b1=1,k1=1,d2=1,k2=1"],
[tstep,0.1],[nsteps,250],[trajectory_at,4,1],
[direction,forward],[x,0,10],[y,0,10]);
la solution forme un cycle
```

À partir du point $(12, 4)$

```
 plotdf([b1*x-k1*x*y,k2*x*y-d2*y],[x,y],
[parameters,"b1=1,k1=1,d2=1,k2=1"],
[tstep,0.1],[nsteps,10000],[trajectory_at,10,4],
[direction,forward],[x,0,12],[y,0,12]);
```

elle s'enroule sur un cycle limite. Le point $(1, 1)$

```
 plotdf([b1*x-k1*x*y,k2*x*y-d2*y],[x,y],
[parameters,"b1=1,k1=1,d2=1,k2=1"],
[tstep,0.1],[nsteps,10000],[trajectory_at,1,1],
[direction,forward],[x,0,2],[y,0,2]);
```

est stationaire.

Question 3 *Dans le cas* $b1 = k1 = d2 = k2 = 1$ *et à partir de la position initiale* $(4, 1)$*, intégrer directement le système en utilisant la méthode d'Euler explicite. Comment varie la solution en fonction du pas de temps en choisissant l'instant initial à 0 et final à 7 ? Comment varie cette solution sur des temps longs (temps final à 20 et essayer un nombre de pas de 1000 puis 1000) ?*

Les fonctions à créer

```
UnPasEulerExplicite(x0,h):=block([x:x0[1],y:x0[2]],[x+h*x*(1-y),y-h*y*(1-x)])$
UPEE:UnPasEulerExplicite$

DesPasFixes(fct,x0,ti,tf,N):=block([h:(tf-ti)/float(N),lst:[x0],i:0,t:ti],
 x0:float(x0),
 for i:1 step 1 thru N do (t:t+h,x0:fct(x0,h),lst:append(lst,[x0])),
 lst)$
DPF:DesPasFixes$
```

La façon de les appeler : ici 500 pas entre 0 et 7

```
plot2d([discrete,DPF(UPEE,[4,1],0,7,500)]);
```

On trouve que la solution entre 0 et 7 se détériore pour des pas de temps trop long ; et la solution entre 0 et 20 est une spirale divergente quelque soit le pas de temps.

Question 4 *Mêmes questions avec la méthode d'Euler implicite. On donne la fonction qui effectue un pas*

```
UnPasEulerImplicite(x0,h):=block([x:x0[1],y:x0[2]],[
((h+1)*sqrt(h^2*y^2+(2*h^2*x-2*h^3+2*h)*y+h^2*x^2+(2*h^3-2*h)*x+h^4-2*h^2+1)
 +(-h^2-h)*y+(h^2+h)*x+h^3+h^2-h-1)
 /(h*sqrt(h^2*y^2+(2*h^2*x-2*h^3+2*h)*y+h^2*x^2+(2*h^3-2*h)*x+h^4-2*h^2+1)
  +h^2*y+h^2*x+h^3-h),
(sqrt(h^2*y^2+(2*h^2*x-2*h^3+2*h)*y+h^2*x^2+(2*h^3-2*h)*x+h^4-2*h^2+1)
 +h*y+h*x+h^2-1)
 /(2*h^2+2*h)]);
UPEI:UnPasEulerImplicite;
```

(elle a été fabriquée à partir de maxima en résolvant le système

```
sol:solve([(x1-x)/h=x1*(1-y1),(y1-y)/h=-y1*(1-x1)],[x1,y1]);
taylor(sol[2],h,0,1);
grind(ratsimp(subst(sol[2],y1)));
```

)

la méthode d'appel est toujours la même

```
plot2d([discrete,DPF(UPEI,[4,1],0,7,500)]);
```

Pour la solution entre 0 et 7, on trouve que la méthode est plus robuste pour les pas de temps longs ; pour la solution entre 0 et 20, elle forme une spirale convergente.

Question 5 *Mêmes questions avec la méthode de Crank-Nicholson. On donne la fonction qui effectue un pas*

```
UnPasCrankNicholson(x0,h):=block([x:x0[1],y:x0[2]],[
-((h*x-2*h-4)*sqrt(4*h^2*y^2+(8*h^2*x-4*h^3+16*h)*y+4*h^2*x^2+(4*h^3-16*h)*x
                              +h^4-8*h^2+16)
 +(2*h^2*x+4*h^2+8*h)*y+2*h^2*x^2+(h^3-4*h^2-12*h)*x-2*h^3-4*h^2+8*h+16)
 /(h*sqrt(4*h^2*y^2+(8*h^2*x-4*h^3+16*h)*y+4*h^2*x^2+(4*h^3-16*h)*x+h^4-8*h^2
                    +16)
  +2*h^2*y+2*h^2*x+h^3-4*h),
(sqrt(4*h^2*y^2+(8*h^2*x-4*h^3+16*h)*y+4*h^2*x^2+(4*h^3-16*h)*x+h^4-8*h^2+16)
 -h^2*y+2*h*x+h^2-4)
 /(h^2+2*h)
]);
UPCN:UnPasCrankNicholson;
```

(qui a été, de même que pour la méthode implicite, fabriquée à partir de

```
sol:solve([(x1-x)/h=(x+x1)/2*(1-(y+y1)/2),(y1-y)/h=-(y+y1)/2*(1-(x+x1)/2)],[x1,y1]);
taylor(sol[2],h,0,1);
grind(ratsimp(subst(sol[2],y1)));
```

)

```
plot2d([discrete,DPF(UPCN,[4,1],0,7,500)]);
```

La solution entre 0 et 7 est bien plus robuste que les précédentes et celle entre 0 et 20 forme un cycle presque parfait pour (presque) tous les pas de temps. C'est normal parce qu'il s'agit d'une méthode implicite d'ordre 2.

Question 6 *Mêmes questions avec la méthode de Runge-Kutta d'ordre 4 ordinaire (on pourra utiliser "rk" du package "dynamics").*

```
load("dynamics");

plot2d([discrete,map(lambda([u],[u[2],u[3]]),rk([x*(1-y),-y*(1-x)],[x,y],[4,1],[t,0,7,0.
```

La méthode entre 0 et 7 est très robuste ; et, de même que la méthode de Crank-Nicholson, la solution entre 0 et 20 forme un cycle presque parfait pour (presque) tous les pas de temps. C'est normal parce qu'il s'agit d'une méthode d'ordre 4, même si elle n'est pas implicite.

Question 7 *Mêmes questions avec la méthode emboîté de Dormand et Prince d'ordre 5 et 4.*
déjà reprendre les

```
dopri54(x0,fct,h):=
block([a21:1/5,
       a31:3/40,a32:9/40,
       a41:44/45,a42:-56/15,a43:32/9,
       a51:19372/6561,a52:-25360/2187,a53:64448/6561,a54:-212/729,
       a61:9017/3168,a62:-355/33,a63:46732/5247,a64:49/176,a65:-5103/18656,
       a71:35/384,a72:0,a73:500/1113,a74:125/192,a75:-2187/6784,a76:11/84,
       b1:35/384,b2:0,b3:500/1113,b4:125/192,b5:-2187/6784,b6:11/84,b7:0,
       bc1:5179/57600,bc2:0,bc3:7571/16695,bc4:393/640,bc5:-92097/339200,bc6:187/2100,bc
       x01,x02,x03,x04,x05,x06,x07,
       f01,f02,f03,f04,f05,f06,f07,
```

```
       x1,xt1
       ],
x01:x0,
f01:fct(x01),
x02:x0+h*a21*f01,
f02:fct(x02),
x03:x0+h*(a31*f01+a32*f02),
f03:fct(x03),
x04:x0+h*(a41*f01+a42*f02+a43*f03),
f04:fct(x04),
x05:x0+h*(a51*f01+a52*f02+a53*f03+a54*f04),
f05:fct(x05),
x06:x0+h*(a61*f01+a62*f02+a63*f03+a64*f04+a65*f05),
f06:fct(x06),
x07:x0+h*(a71*f01+a72*f02+a73*f03+a74*f04+a75*f05+a76*f06),
f07:fct(x07),
x1:x0+h*(b1*f01+b2*f02+b3*f03+b4*f04+b5*f05+b6*f06+b7*f07),
xt1:x0+h*(bc1*f01+bc2*f02+bc3*f03+bc4*f04+bc5*f05+bc6*f06+bc7*f07),
[x1,xt1]
)$
```

et (attention, il faut prendre celle qui est approprié à un système différentiel)

```
integre_edo(fct,x0,t0,tf,h,tol):=block([x1,tx1,X:[[t0,x0]],ok],
while (t0 < tf) do
(if (t0+h > tf) then h:tf-t0,
 tx1:float(dopri54(x0,fct,h)),
 x1:tx1[1],
 tx1:tx1[2],
 ok:true,
 map(lambda([u,v],if (abs(u)<v) then ok:true else ok:false),tx1-x1,tol),
 if ok then (t0:t0+h,x0:x1,X:append(X,[[t0,x0]]),h:1.2*h)
 else (h:h/2)),
X)$
```

puis le fonctionnement est

```
f(xy):=block([x:xy[1],y:xy[2]],[x*(1-y),-y*(1-x)]);
eps:0.00001;
plot2d([discrete,map(lambda([u],[u[2][1],u[2][2]]),
integre_edo(f,[4,1],0,7,1,[eps,eps]))]);
```

la conclusion est que la solution est très bonne quoi qu'il arrive ; le tracé affine par morceau est un peu trompeur, comme dopri54 est une méthode d'ordre 5 (ou 4), il faudrait une interpolation par des polynômes de cet ordre pour le tracé et donc un post-processing plus sophistiqué que celui-ci.

Question 8 *Comparer la méthode dopri54 avec la méthode de Runge-Kutta d'ordre 4. Combien faut-il prendre de pas de temps dans Runge-Kutta d'ordre 4 pour obtenir une précision analogue à celle de dopri54 (tolérance* 10^{-5} *en* x *et* y*) ?*

```
 sol7do:block([sol:integre_edo(f,[4,1],0,7,1,[eps,eps])],
 [length(sol),last(sol)]);
 sol7do:flatten(sol7do);
```

```
block([sol:rk([x*(1-y),-y*(1-x)],[x,y],[4,1],[t,0,7,0.1])],
sol7rk:flatten([length(sol),last(sol)]),
disp(["dopri",sol7do],["rk4  ",sol7rk],["diff ",sol7rk-sol7do]));
```

dopri54 demande bien moins de pas que Runge-Kutta pour une précision équivalente. Mais c'est à relativiser parce qu'il y a plus d'évaluation de fonction dans un pas de dopri que dans un autre de rk...